AD A0 65535

Bolt Beranek and Newman Inc.

LEVEL III

~~F02~~ SC

13

A056838

Report No. 3963

# Research in Natural Language Understanding

Quarterly Progress Report No. 4, 1 June 1978 to 31 August 1978

Annual Report, 1 September 1977 to 31 August 1978

Prepared for:
Advanced Research Projects Agency

79 03 07 0

6

RESEARCH IN NATURAL LANGUAGE UNDERSTANDING •

9 Quarterly Technical Progress Report, No. 4

1 June ~~1978~~ — 31 August ~~1978~~ 78.

Annual Report

1 September 1977 - 31 August 1978

14 BBN-3963

15

ARPA Order No. 3414

Program Code No. 8D30

Name of Contractor:
Bolt Beranek and Newman Inc.

Effective Date of Contract:
1 September 1977

Amount of Contract:
$712,572

Contract No. N00014-77-C-0378,

VV ARPA Order-3414

~~Contract Expiration Date:~~
31 August 1979

Short Title of Work:
Natural Language Understanding

10

Principal Investigator:
Dr. William A. / Woods
(617) ~~491-1850 x361~~

Ronald J. / Brachman

~~Scientific Officer:~~
Gordon D. Goldstein

11 31 Aug 78

12 82p.

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 3414

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-77-C-0378.

060100

79 03 07 1 2

Unclassified.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>BBN Report No. 3963 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>RESEARCH IN NATURAL LANGUAGE UNDERSTANDING<br>Q.T.P.R. No. 4, 6/1/78 - 8/31/78<br>Annual Report, 9/1/77 - 8/31/78 | | 5. TYPE OF REPORT & PERIOD COVERED<br>Quarterly Progress Report<br>Annual Report |
| | | 6. PERFORMING ORG. REPORT NUMBER<br>Report No. 3963 |
| 7. AUTHOR(s)<br>W.A. Woods and R.J. Brachman | | 8. CONTRACT OR GRANT NUMBER(s)<br>N00014-77-C-0378 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Bolt Beranek and Newman Inc.<br>50 Moulton Street<br>Cambridge, MA 02138 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Office of Naval Research<br>Dept. of the Navy<br>Arlington, VA 22217 | | 12. REPORT DATE<br>31 August 1978 |
| | | 13. NUMBER OF PAGES<br>77 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>Unclassified |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Semantic networks, knowledge representation, cognitive science, artificial intelligence, natural language understanding, natural language conceptual structure, ATN grammars, natural language grammars, perceptual automata, structured inheritance networks, KLONE, memory models, property inheritance, augmented transition networks, parsing, syntactic-semantic interactions.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report contains an annual summary of BBN's ARPA-sponsored natural language understanding project and two Technical Notes: "KLONE's Progress," by R.J. Brachman, and "Generalizations of ATN Grammars," by W.A. Woods. The Annual summary includes a list of publications of the project and an overview of the first year's work. The first Technical Note focuses on the progress in the development of the KLONE knowledge representation system, an implementation of Brachman's Structured

cont'd.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (cont'd.)

Inheritance Networks. The second describes some generalizations of the notion of ATN grammar, including Cascaded ATN's (a sequence of nondeterministic ATN transducers with each successive ATN taking its input from the output of the previous), Generalized Transition Networks (a generalization of the notion of ATN to handle arbitrary structured input rather than only linear strings), and an interpretation of KLONE networks as a kind of ATN with inheritance.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

## TABLE OF CONTENTS

## Annual Summary

W. A. Woods

During the past year, BBN's ARPA project on natural language understanding has been pursuing a number of diverse problems, all of which are major bottlenecks to significant advances in natural language understanding by machine. These problems impose significant limitations on the kinds of intelligent support for complex decision making that can be gained from computerized knowledge bases. The directions that we have been pursuing include:

- representation of complex natural language concepts.
- representational structures that facilitate use of such knowledge.
- algorithms for recognition of instances of complex structured concepts.
- parsing algorithms for interfacing to such concept recognition algorithms.
- the relationship between such parsing algorithms and the recognition algorithms themselves.
- formal properties of parsing and situation recognition algorithms.
- formal semantics and epistemological foundations of knowledge representation structures.
- application of such structures and algorithms to practical problems of command and control decision support.
- concepts for parallel machine architectures to support such applications.

In addition, we have continued to make advances in the codification and formalization of rules of human use of natural language for communication, most notably in the formalization of evoked entities that become available for anaphoric reference in discourse.

This work can be broken down into three major areas:

1. Knowledge Representation
2. Syntactic and Semantic Processing
3. Fast Symbolic Algorithms

Our activities in knowledge representation and its use for situation recognition have centered around the refinement and use of structured inheritance networks [Brachman, 1978a,b] for representing complex concepts, ATN grammars [Woods, 1970] for parsing natural language sentences, and relationships between the two. We have been gradually uncovering deep similarities between the parsing functions of ATN grammars and the recognition and understanding functions of a conceptual taxonomy expressed as a structured inheritance network. (Some of this is described in more detail in the Technical Notes in this report.)

During the past year, we have implemented a computerized interpretation package for a particular version of structured inheritance networks (KLONE), and have successfully used that representation system to develop a taxonomy of kinds of syntactic structures as well of the concepts that they can signify. Further, we have developed a method for representing the connections between the two in the form of "interpretation cables" which can be used to develop incremental semantic interpretations in the course of parsing. We have also developed and refined a parsing system (RUS) that provides close and efficient interaction between a general syntactic processor and a semantic-pragmatic interpreter.

Experience with the RUS grammar, and its coupling to the KLONE network taxonomies has reinforced an emerging understanding of the utility of a kind of parsing automaton which I have called "cascaded ATN's", a sequence of ATN transducers each of which takes as input the output of the previous transducer. Some theoretical insight into the computational advantages of such cascades has been emerging, and several formal results on the computational power of ATN grammars have been developed.

Also in the area of syntactic/semantic processing, we have:

- Completed an extensive theoretical study of mechanisms of pronominal and anaphoric reference in English, culminating in a Ph.D. thesis at Harvard University [Webber, 1978], which appears likely to establish a paradigm for the linguistic investigation of anaphora.

- Developed a new control strategy for ATN parsers which, together with appropriate grammar structuring, results in an "almost-deterministic" parsing strategy that combines many of the best features of Marcus' deterministic parser [Marcus, 1978] and the above mentioned close-coupled ATN parser/interpreter.

- Written a report on semantics and quantification in natural language question answering, including a retrospective analysis of the techniques of the LUNAR system and their relative advantages and disadvantages. This study led to an implementation of a powerful and general quantification facility in the RUS system, and an investigation of new types of quantifier representation using the SI-net notation.

- Developed some theoretical results on the power of ATN parsers and the advantages of close-coupled systems of parsing and interpretation.

In the area of fast symbolic algorithms, we have identified a general problem of situation recognition as a problem of major importance across many areas of artificial intelligence, including parsing and interpreting natural language, interpretation of visual scenes, monitoring for alerting conditions in large data bases, rule selection in large systems of production rules, and special case recognition in problem solving and mechanical inference systems. We have been concentrating on developing potentially fast algorithms and efficiency techniques for this problem, and have identified and partially developed two major techniques:

- Hypothesis-Factoring, which operates to cut down combinatoric enumeration, and

- Marker-Passing Algorithms, which facilitate the exploitation of specialized parallel processing architectures.

Specifically, we have:

- Developed techniques for minimizing the combinatorics of hypothesis enumeration in ATN parsing, and

- Developed an abstract parallel machine architecture and a corresponding marker passing algorithm for semantic interpretation of English sentences. Although this algorithm is still under active development, an early version of it has been documented in a quarterly report.

During the coming year, we plan to:

- Continue our investigation of knowledge representation issues - especially in the areas of plurality and sets, mutual exclusion and exhaustiveness, temporal history and change in large data bases, inheritance mechanisms, and uses of meta knowledge.
- Implement the anaphoric reference system described above and integrate it with the RUS parsing system.
- Assemble an integrated system to understand natural language display manipulation instructions.
- Continue to explore the close coupling between syntactic and semantic processing - especially for resolving ambiguity and vagueness.
- Implement a facility for speech act interpretation and plan recognition.
- Continue the theoretical investigations of fast algorithms for situation recognition and related processes.
- Continue our investigation of parallel architectures and algorithms for situation recognition in general, and for natural language understanding in particular.

## Publications

During the past year, the following reports were published or accepted for publication:

**Semantics and Quantification in Natural Language Question Answering**

W. A. Woods
BBN Report No. 3687

November 1977

This paper is concerned with the semantic interpretation of natural English sentences by a computerized question-answering system, and specifically with the problems of interpreting and using quantification in such systems. These issues are presented and discussed from the perspective of four different natural language understanding systems with which the author has been involved. The presentation includes the process of semantic interpretation, the nature and organization of semantic interpretation, the nature and organization of semantic interpretation rules, a notation for representing semantic interpretations (the meaning representation language), the semantics of that notation, and the generation and scoping of quantifiers. Also discussed are a variety of loose ends, open questions, and directions for future research. Particular attention is given to the interaction of syntactic, semantic (and pragmatic) information.

**On The Epistemological Status of Semantic Networks**

Ronald J. Brachman
BBN Report No. 3807*

April 1978

* To appear in **Associative Networks - The Representation and Use of Knowledge in Computers**. Nicholas V. Findler, ed. New York: Academic Press, 1978.

This paper examines in detail the history of a set of network-structured formalisms for knowledge representation - the so-called, "semantic networks". Semantic nets were introduced around 1966 as a representation for the concepts underlying English words, and since then have become an increasingly popular type of language for representing concepts of a widely varying sort. While these nets have for the most part retained their basic associative nature, their primitive representational elements have differed significantly from one project to the next. These differences in underlying primitives are symptomatic of deeper philosophical disparities, and I discuss a set of five significantly different "levels" at which networks can be understood. One of these levels, the "epistemological", or "knowledge-structuring, level, has played an important implicit part in all previous notations, and is here made explicit in a way that allows a new type of network formalism to be specified. This new type of formalism accounts precisely for operations like individuation of description, internal concept structure in terms of roles and interrelations between them, and structured inheritance. In the final section of the paper, I present a brief sketch of an example of a particular type of formalism ("Structured Inheritance Networks") that was designed expressly to treat concepts as formal representational objects. This language, currently under development, is called "KLONE", and it allows the explicit expression of epistemological level relationships as network links.

## A Formal Approach to Discourse Anaphora

Bonnie Lynn Webber
BBN Report No. 3761

May 1978

Extended natural language communication between a person engaged in solving a problem or seeking information and a machine providing assistance requires the machine to be able to deal with anaphoric language in a perspicuous, transportable non-ad hoc way. This report takes the view that dealing with anaphoric language can be decomposed into two complementary tasks: (1) identifying what a text potentially makes available for anaphoric reference and (2) constraining the candidate set of a given anaphoric expression down to one possible choice. The second task has been called the "anaphor resolution" problem and, to date, has stimulated much research in psychology and artificial intelligence natural language understanding.

The focus of this report is the first task - that of identifying what a text makes available for anaphoric reference and how it does so. Evidence is given to back up two strong claims:

1. None of the three types of anaphoric expressions that I have studied - definite anaphora, "one"-anaphora and verb phrase deletion - can be understood in purely linguistic terms. That is, none of them can be explained without stepping out of the language into the conceptual model each participant is synthesizing from the discourse.

2. On the other hand, if a discourse participant does not assign to each new utterance in the discourse a formal representation in which, inter alia,

   a. quantifiers are indicated, along with their scopes;
   b. main clauses are distinguished from relative clauses and subordinate clauses;
   c. clausal subjects are separated from clausal predicates;

   then s/he will not be able to identify all of what is being made available for anaphoric reference.

Building on these claims, I show that there is an intimate connection between such a formal sentential analysis and the synthesis of an appropriate conceptual model of the discourse. The computational implications of this research are discussed, primarily in terms of possible implementations within current levels of technology.

## Description Formation and Discourse Model Synthesis

Bonnie Lynn Webber

July 1978

This paper starts from the point of view that a common objective of discourse is to direct a listener in the construction of a model of some particular or general situation. Its concern is how the explicit data of the discourse provides material for the model synthesis process. In particular, it shows how (1) indefinite noun phrases are associated with the evocation of new entities into the model ("discourse entities") and (2) how those new discourse entities will initially be described. The claim is that such an initial description is critical to both model synthesis and anaphor resolution since it allows the listener to reason appropriately about entities in order to assign them to appropriate roles vis-a-vis his or her higher-level expectations.

## Taxonomic Lattice Structures for Situation Recognition

W.A. Woods

July 1978

This paper discusses issues in the representation of knowledge for an intelligent computer assistant to a human decision maker. A substantial portion of the knowledge base of such a system will consist of rules of the form "if <situation description> do <action>", where situation descriptions are characterizations of classes of situations that the machine could be in. A taxonomic lattice is a structure that organizes such situation descriptions into an inheritance structure that permits information to be stored in its most general form and yet still be triggered by any situation to which it applies. This lattice serves as a "coat rack" upon which various pieces of advice may be hung, and is accessed by the machine in order to find and activate advice that applies to the current situation. The process of situation recognition is similar to the process of parsing in that only as a result of its recognition is a situation transformed from a collection of unrelated events and conditions into a perception of a structured concept. Two methods are discussed for efficient use of a taxonomic lattice for situation recognition - factored knowledge structures, which merge common parts of alternative hypotheses, and markable classification structures, whose nodes serve as rendezvous points where "footprints" from various constituents can meet to detect coincidences.

# Research in Natural Language Understanding

Quarterly Technical Progress Report No. 1
1 September 1977 to 30 November 1977

TABLE OF CONTENTS

## Research in Natural Language Understanding

Quarterly Progress Report No. 2
1 December 1977 to 28 February 1978

TABLE OF CONTENTS

## Research in Natural Language Understanding

Quarterly Progress Report No. 3
1 March 1978 to 31 May 1978

TABLE OF CONTENTS

Related Reports:

**A Structural Paradigm for Representing Knowledge**

R. J. Brachman
BBN Report No. 3605

May 1978

This report presents an associative network formalism for representing conceptual knowledge. While many similar formalisms have been developed since the introduction of the "semantic network" in 1966, they have often suffered from inconsistent interpretation of their links, lack of appropriate structure in their nodes, and general expressive inadequacy. In this paper, we take a detailed look at the history of these "semantic" nets and begin to understand their inadequacies by examining closely what their representational pieces have been intended to model. Based on our analysis, we present a new type of network - the "Structured Inheritance Network" (SI-NET) - designed to circumvent common expressive shortcomings. We acknowledge "concepts" to be formal representational objects and keep "epistemological" relationships between formal objects distinct from conceptual relations between the things that the formal objects represent. The notion of an **epistemologically explicit** representation language is introduced to account for this distinction, and SI-Nets are offered as a particular candidate.

The Structured Inheritance formalism that we present takes a concept of **functional roles** tied together by a **structuring gestalt**. Generic concepts, describing potentially many individuals, have as their parts generic "'dattr' descriptions", which capture information about the functional role, number, criteriality, and nature of potential role fillers and "structural conditions", which express explicit relationships betwe[illegible] the potential role fillers, and give the functional roles their [illegible]eanings. Individual concepts have explicit binding structures ("Instantiated dattrs") which indicate an individual's fillers for its roles: the individual's roles are inherited from a generic concept, in terms of which it is described. Details of the representation are elaborated, including explicit role and role-filler inheritance rules. The language is then applied to two task domains: 1) the understanding of two-word nominal compounds (e.g., "computer science", "arm chair", "hockey stick"), for which we present a conceptual analysis that uses only two basic structuring techniques to explain an extensive set of compound types; we also present a new account of nominalization, based on structured inheritance; and 2) knowledge about a complex message-processing program that is implemented on several ARPA Network hosts; we attempt to account for the structure of objects in the "Hermes" program, its commands, and the interaction of the commands and objects.

In addition to detailing these uses of the structural paradigm, we review carefully its relationship to three other current representation languages - KRL, FRL, and MDS. The surface notation, underlying data structures, and deeper epistemological import of each of these languages is examined and compared with the others.

**KLONE Reference Manual**

R.J. Brachman
E. Ciccarelli
N. Greenfeld
M. Yonke

July 1978

KLONE is being developed to be an epistemologically-explicit language for representing conceptual knowledge and structured inheritance; this manual provides user documentation for the current state of the INTERLISP implementation. Documented are: types of KLONE entities and relationships; procedural and data attachment; conceptual "meta-description" of KLONE entities; implementation naming conventions; all user-accessible KLONE primitives.

## KLONE's Progress

R.J. Brachman

This year was spent laying the foundation for the complete natural language system that we plan to build in the second year. We concentrated on two major components - the parser and the knowledge representation - working on them independently and jointly. We designed the "abstract machines" for these two parts of our system, implemented and tested those designs, designed and implemented an interface to allow natural language information to be projected into the representation system, and built some experimental systems to help us get a better understanding of the potential overall complexity of a complete system.

Our knowledge representation language, KLONE, has undergone several cycles of revision during the course of the year. KLONE is directly descended from work reported in [Brachman, 1978b], and since its adoption into this project has become a sophisticated, state-of-the-art representation system. There are still several open issues to be resolved in the abstract design of the language, but the current conception is virtually all implemented and usable. The current implementation work consists of completing some more recent design features, and finishing a complete set of "Structural Description" (SD) manipulation functions. The system as described in [Woods & Brachman, 1978] has been in use for about half of the year.

Work on our abstract conception of KLONE continues at a lively pace, although we believe that we are gradually obtaining a sense of closure on many representation issues. Each round of design/implementation/use will, of course, bring new information to bear on the original design, but the number of unresolved issues is definitely diminishing.

During the course of the year, we have come to a much clearer understanding of **inheritance** in classificatory network structures. We have begun to think of such relations between Concepts as "**Cables**", which themselves have structure and can be talked about (e.g., meta-described). The Cable notion* helps to substantially solidify one of the principal insights of [Brachman, 1978b] - that there is no simple "ISA" link, but instead, an inheritance connector must allow access to subparts of the description of the Concept. Subpart connections between sub- and superConcepts are not independent of the connections between the Concepts as wholes.

The inter-Role connections that make up the bulk of the Cable have also become better understood, and we have re-implemented the

* Our cable metaphor comes more or less directly from the work of Brian Smith [1978].

Role facet inheritance functions. The "Mods" primitive means that the subRole is essentially overlaid on top of its superRole, with only unmodified facets remaining visible. Thus, there is only one Role to be considered as a Mods chain is descended - the subRole is virtually the very same Role as the one it modifies. The "Diffs" connector, on the other hand, indicates that a Role spawns distinguishable subRoles. Each subRole inherits its name and other applicable subparts from its parent, but is considered a "real" Role unto itself. The more specific Role is in a sense subsumed by its parent, but it is not indistinguishable from that parent. This clearer understanding of the meaning of the role inheritance primitives has made their implementation easier, and has also served to facilitate the adjudication of putative conflicts in situations of multiple inheritance.

The idea of a "ParaIndividual", developed during the course of this year, has also helped clarify the meaning of the SD's, and has put us in a better position to solve the puzzle of recursive descriptions in a declarative representation language. It has also highlighted a strong similarity between SD's and Roles - the Value Restriction of a Role is almost interpretable as a ParaIndividual: it has existence local to the Concept in which it appears, and its value in each Individuator is contextually determined by that Individuator. ParaIndividuals have been implemented, and we are in the midst of investigating their utility and meaning.

In the last quarter of this year, we produced a new proposal for interpretive hooks (ihooks), a proposal for allowing SD's to have names, a set of proposals regarding the representation of quantification, and a cleaner separation between individuals and descriptions. We also embarked on an investigation into the explicit representation of descriptions of sets. On a larger scale, we also designed and initially implemented an integrated KLONE/parser system for understanding simple sentences about the graphics domain.

As we began to make extensive use of he ihook facility for attaching procedures, we found it deficient in a number of ways. First, the simple BEFORE-<procedure>/AFTER-<procedure> dichotomy did not provide a rich enough set of invoking situations. We needed some procedures to be invoked after KLSpecializeRole and after KLEstablishAsSpecializer, for example. That is, we wanted to activate an attached procedure upon the specialization of a Role, regardless of which function produced the specialization. Thus, we required a better taxonomy of invoking situations. Further, "BEFORE" and "AFTER" were not quite right - the main intent of a "BEFORE" hook is to allow a set of preconditions to be tested before the procedure is actually invoked; and "AFTER" hooks have two uses: as postcondition-checkers, so that an undesirable effect of an attached procedure could be aborted; or as "consequent theorems" (a la PLANNER), such as "WHEN-FILLED". Finally, all procedures from all parents were inherited indiscriminately - no procedure from a parent Concept could be "turned off" in one of its descendants.

This set of difficulties led us to redesign the ihook feature. We decided that instead of two basic situation types, there should be three: **PRE-**, **POST-**, and **AFTER-**. The first two will be invoked right before and right after the main function body (of the interpreter primitive), respectively, and will be treated as predicates. If a "PRE-" or "POST-" condition fails, then the interpreter primitive will be aborted, without any effect. An "AFTER-" procedure will run after successful completion of the function body, and its effect will not be of consequence to the function. In addition, ihook situations will be more general than simple function names. A taxonomy of situations, like "INDIVIDUATION" and "SPECIALIZATION", will allow the same procedure to be invoked by distinct, but conceptually similar, functions. Finally, we will allow a user to attach to an ihook an arbitrary keyword (to symbolize for him the "intent" of the hook). Procedures will be inherited as before, but only the most local will be executed for any given keyword. This new facility will give us a tremendous repertoire of power to experiment with.

As mentioned, we have added "SDNames", which will work for SD's much as RoleNames work for Roles. This will allow us to address segments of a Concept's internal structure directly, and will allow selective building of new SD's in the conceptual structure of our language understanding system.

We spent much of the final quarter of this year discussing the handling of natural language quantification in KLONE. While much depends on the particular use of KLONE chosen for the language interface, one observation is worth mentioning in and of itself. After investigating a number of natural language phenomena, we decided that it made sense to separate the "lexical", or propositional, content of a sentence from its quantificational content [see Van Lehn, 1978]. The lexical portion of the sentence explains the type of entity doing the type of action to a type of object, etc. The quantification structure can then be superimposed on top of the lexical material to explain how many actions there were, how many actors, etc., and the nature of the map from actor onto action, etc. For example, in "Every boy kissed three girls", the verb and the two nouns serve to relate that some kissing activity(ies) transpired between some boy(s) and some girl(s). On top of that, the quantificational structure leaves the total number of participants open, (presumably, there is some implied set over which the universal ranges) but says that each kissing event had one object (a girl) and one agent (a boy), and that each boy mapped onto three kissing events. We are actively pursuing a clear and adequate representation scheme for relating lexical and quantificational import. The status of this enterprise will be reported in subsequent QPR's.

# Generalizations of ATN Grammars

W. A. Woods

## 1. Introduction

ATN grammars, as presented in Woods [1970] are a form of augmented pushdown store automata, augmented to carry a set of register contents in addition to state and stack information and to permit arbitrary computational tests and actions associated with the state transitions. Conceptually, an ATN consists of a network of states with connecting arcs between them. Each arc indicates a kind of constituent that can cause a transition between the states it connects. The states in the network can be conceptually divided into "levels" corresponding to the different constituents that can be recognized. Each such level has a start state and one or more final states. Transitions are of three basic types, as indicated by three different types of arc. A WRD (or CAT) transition corresponds to the consumption of a single word from the input string, a JUMP transition corresponds to a transition from one state to another without consuming any of the input string, and a PUSH transition corresponds to the consumption of a phrase parsed by a subordinate invocation of some level of the network to recognize a constituent.

ATN's have the advantage of being a class of automata into which ordinary context-free phrase structure and "augmented" phrase structure grammars have a straightforward embedding, but which permit various transformations to be performed to produce grammars that can be more efficient than the original. Such transformations can reduce the number of states or arcs in the grammar or can reduce the number of alternative hypotheses that need to be explicitly considered during parsing. (Some transformations tend to reduce both, but in general there is a tradeoff between the two). Both kinds of efficiency result from a principle that I have called "factoring", which amounts to merging common parts of alternative paths in order to reduce the number of alternative combinations explicitly enumerated. The former results from factoring common parts of the grammar to make the grammar as compact as possible, while the latter results from arranging the grammar so as to factor common parts of the hypotheses that will be enumerated at parse time. The former promotes ease of human comprehension of the grammar and should facilitate learning of grammars by machine. The latter promotes efficiency of run time execution. I will refer to the former as "conceptual factoring" and the latter as "hypothesis factoring".

Many of the same factoring principles that apply to ATN grammars of the ordinary kind can be applied to other problems not directly interpretable as consuming elements from sequences of symbols. In this paper, I will present some generalizations of the notion of ATN grammar that capitalize further on the principle of factoring and that are applicable to a much more diverse set of situations. The first is obtained by generalizing ATN's from simple parsers to transducers by the the addition of an output operation ("TRANSMIT") which can be executed on arcs, followed by the construction of a parsing automaton from a cascade of such ATN transducers. The resulting automaton, which I call an "ATN cascade", gains a factoring advantage from merging together common computations at early stages of the cascade.

Cascaded ATN's are analogous to certain state decomposition characterizations of finite state machines and carry many of the advantages of such state decomposition into the domain of more general recognition automata. The normal decomposition of natural language description into levels of phonology, lexicon, syntax, semantics, and pragmatics, can be viewed as a cascade of ATN transducers - one for each of the individual levels. Viewing natural language understanding as parsing with such a cascade has computational advantages and also provides an efficient, systematic framework for characterizing the relationships between different levels of analysis due to conceptual factoring. The factoring advantages of cascade decompositions can thus serve as a partial explanation of why such a componential description of natural language understanding has arisen.

A second generalization of ATN's lifts the implicit assumptions that the input is a sequence of symbols and permits the application of similar factoring and optimization techniques to the general case of recognition automata acting on a generalized "perceptual field". Such generalized transition networks (GTN'S) have potential applications in scene analysis, acoustic phonetic analysis of speech, medical diagnosis, discourse analysis, and data base monitoring for "alerting" capabilities. Generalized tiansition networks thus lift the notion of "grammar" away from the limited conception of a set of rules characterizing well-formed sequences of words in sentences. Rather, they are capable of characterizing arbitrary classes of structured entities.

Finally, I will present an interpretation of structured inheritance networks [Brachman, 1978b; Brachman & Woods, 1978] as generalized transition networks and discuss the new perspectives introduced by the addition of a concept of inheritance to the concept of grammar.

## 2. Factoring in ATN's and PSG's

As discussed above, the principle of factoring involves the merging of common parts of alternative paths through an ATN or similar structure in order to minimize the number of combinations. This can be done either to reduce the size of the grammar or to reduce the number of alternative hypotheses considered at parse time. **Conceptual** factoring attempts to reduce the size of the grammar by minimizing the number of places in the grammar where the same or similar constituents are recognized. Frequently such factoring results from "hiding" some of the differences between two paths in registers so that the paths are otherwise the same and can be merged. For example, in order to represent number agreement between a subject and a verb, one could have two distinct paths through the grammar - one to pick up a singular subject and correspondingly inflected verb, and one to pick up a plural subject and its verb. By keeping the number of the subject in a register, however, one can merge these two paths so there is only one push to pick up the subject noun phrase and one push to pick up the main verb.

In other cases, conceptual factoring results from merging common initial, final, and/or medial sequences of paths across a constituent that are not the same, but which share subsequences. For example, an interrogative sentence can start with an auxiliary verb followed by the subject noun phrase, while a declarative can start with a noun phrase followed by the auxiliary. In either case, however, the subsequent constituents that can make up the sentence are the same and the grammar paths to recognize them can be merged. Moreover, in either case there can be initial prepositional phrases before either the subject or the auxiliary and again these can be merged. When one begins to represent the details of supporting auxiliaries that are present in interrogatives but not in declaratives, the commonalities these modalities have with imperatives, and the interaction of all three with the various possibilities following the verb (depending on whether it is transitive or intransitive, takes an indirect object or complement, etc.), this kind of factoring becomes increasingly more important.

In ordinary phrases structure grammars (PSG's), the only mechanism for capturing the kinds of merging discussed above is the mechanism of **recursion** or "pushing" for constituent phrases. In order to capture the equivalent of the above merging of commonality between declaratives and interrogatives, one would have to treat the subject-auxiliary pair as a constituent of some kind (an organization that is linguistically counter-intuitive). Alternatively, one can capture such factoring in a PSG by emulating an ATN - e.g., by constructing a phrase structure rule for every arc in the ATN and treating the states at the ends of the arc as constituents. Specifically, an arc from s1 to s2 that picks up a phrase p can be represented by a phrase structure rule s1 -> p s2, and a final state s3 can be expressed by an "e rule" s3 -> e (where e represents the "empty string"). In either case, one is forced to introduce a "push" to a lower level of recursion where it is not necessary for an ATN, and to introduce a kind of "constituent" that is motivated solely by principles of factoring and not necessarily by any linguistic criteria of constituenthood.

A phrase structure grammar emulating an ATN as in the above construction will contain all of the factoring that the ATN contains, but will not make a distinction between the state name and the phrase name. Failure to make this distinction masks the intuitions of state transition that lead to some of the ATN optimization transformations and the conceptual understanding of the operation of ATN's as parsing automata. The difference here is a lot like the difference between the way that LISP implements list structure in terms of an underlying binary branching "cons" cell and the way that it is appropriate to view lists for conceptual reasons. For exactly the same kinds of reasons, it is appropriate to think of certain sequences of constituents that make up a phrase as sequences of immediate constituents rather than as a right-recursive nest of binary branching phrases.

From the perspective of **hypothesis** factoring, the distinction made in an ATN between states that can be recursively pushed to and states that merely mark intermediate stages in the recognition of a constituent sequence permits a distinction between that part of a grammar that is essentially finite state (and hence amenable to certain kinds of optimization) and that which is inherently recursive. This permits such operations as mechanically eliminating unnecessary recursion and performing finite-state optimizations procedures on what remains - see Woods [1969]. These transformations can result in significant gains in parsing efficiency by trading recursion for iteration wherever possible and by minimizing the non-determinism (by hypothesis factoring) in the resulting networks.

The construction given above for emulating an ATN with a PSG can, of course, emulate the same hypothesis factoring optimization that an ATN permits, but its ability to do so depends critically on the use of e-rules for the final states. Most parsers for PSG's, on the other hand, do not permit e-rules, probably because they are highly non-deterministic when applied bottom-up. Unfortunately, the construction that transforms a PSG with e-rules into an equivalent PSG with no e-rules would give up some of the factoring achieved in the ATN emulation when applied to final states that are not obligatorily final (a common occurrence in natural language grammars). Every transition coming into such a state, would effectively be duplicated - once leading to an unambiguously final state (sl -> p), and once forcing subsequent consumption of additional input (s1 -> p s2). It thus appears that as a class of formal automata, ATN's permit a greater flexibility in capturing hypothesis factoring advantages than do conventional PSG's.

As we have discussed them, the principles of conceptual factoring and hypothesis factoring have been motivated by different measures of cost. Nevertheless, many of the factoring transformations that can be applied to ATN's gain a simultaneous efficiency in both dimensions. This is not always the case however. In particular, the transformations that optimally minimize nondeterminism for left-to-right parsing tend to cause an increase in the number of states and arcs in a grammar (unless fortuitous regularity causes a collapsing). Since a major characteristic of the ATN grammar formalism is that it permits the expression of mechanical algorithms for performing hypothesis factoring transformations, it is probably appropriate for grammar writers to devote their attention to conceptual factoring as a grammar writing style, while leaving to various grammar compilation algorithms the task of transforming the grammar into an efficient parsing engine. However, in absence of such compilers, it is always possible within the ATN formalism for a grammar writer to incorporate explicit hypothesis factoring structure into his grammar and to make tradeoffs between the two factoring principles.

## 3. Notation

ATN's are characterized as automata by specifying their computations in terms of instantaneous configurations and a transition function that computes possible successor configurations. As such, they can admit a variety of superficial syntaxes, without changing the essential nature of the automaton. In this paper, I will use a notation that is somewhat more concise and slightly more convenient than the original ATN syntax specified in Woods [1970]. The major change will be a formal distinction between a phrase type and an initial state for recognizing a phrase. (The original ATN specification used the initial state to serve double duty.) Moreover, I will permit a given phrase type to have several distinct initial states and for several phrase types to share some initial states. This permits somewhat greater flexibility in factoring and sharing common parts of different phrase types. The pop arcs of these ATN's will indicate the phrase type being popped, and a given state can be a final state for several phrase types. A BNF specification of the syntax I will use is:

```
<ATN> -> (<machinename> (accepts <phrasetype>*) <statespec>*)
                ;an ATN is a list consisting of a machine name, a
                ;specification of the phrasetypes which it will
                ;accept, and a list of state specifications.
<statespec> -> (<statename> {optional <initialspec>} <arc>*)
<initialspec> -> (initial <phrasetype>*) ;indicates that this state
                ;is an initial state for the indicated phrasetypes.
<arc> -> (<phrasetype> <nextstate> <act>*)  ;a transition that
                                ;consumes a phrase of indicated type.
      -> (<pattern> <nextstate> <act>*) ;a transition that consumes
                               ;an input element that matches a pattern.
      -> (J <nextstate> <act>*) ;a transition that jumps to a new
                                ;state without consuming any input.
      -> (POP <phrasetype> <form>)  ;indicates a final state
                           ;for the indicated phrase type and specifies
                           ;a form to be returned as its structure.
<nextstate> -> <statename> ;specifies next state for a transition.
<pattern> -> ( <pattern>* ) ;matches a list whose elements match
                            ;the successive specified patterns.
          -> <wordlist>     ;matches any word in the list.
          -> &              ;matches any element.
          -> --             ;matches any subsequence.
          -> <form>         ;matches value of <form>.
          -> <<classname>> ;matches anything that has or inherits
                            ;the class name as a feature.
<wordlist> -> {'<word> | '<word>, <wordlist>}
<act> -> (transmit <form>) ;transmit value of form as an output.
      -> (setr <registername> <form>) ;set register to value of form.
      -> (addr <registername> <form>) ;add the value of form to the
                 ;end of the list in the indicated register (assumed
                 ;initially NIL when the register has not been set).
      -> (require <proposition>) ;abort path if proposition is false.
      -> (dec <flaglist>)          ;set indicated flags.
      -> (req <flagproposition>) ;abort path if proposition is false.
      -> (once <flag>) ;equivalent to (req (not <flag>)) (dec <flag>).
<flagproposition> -> <boolean combination of flag registers>
<proposition> -> <form>    ;the proposition is false if the value
                            ;of the form is NIL.
<form> -> !<registername>   ;returns contents of the register.
       -> '<liststructure>  ;returns a copy of a list structure
                 ;except that any expressions preceded by ! are
                 ;replaced by their value and any preceded
                 ;by @ have their value inserted as a sublist.
       -> !c  ;contents of the current constituent register.
       -> !<liststructure>  ;returns value of list structure
                            ;interpreted as a functional expression.
```

A simple example, using the above conventions, is the following grammar:

```
(m (accepts q)
  (s1 (initial q)
    ('a s2 (setr n 1)))
  (s2
    (q s3 (setr n !(1 + !c)))
    (J s3))
  (s3
    ('b s4))
  (s4
    (pop q !n)))
```

This grammar parses a string of n a's followed by n b's and pops the number n.

## 4. Cascaded ATN's

The advantages of having semantic and pragmatic information available at early stages of parsing natural language sentences have been demonstrated in a variety of systems.* Ways of achieving such close interaction between syntax and semantics have traditionally involved writing semantic interpretation rules in 1-1 correspondence with phrase structure rules (e.g., Thompson [1963]), writing "semantic grammars" that integrate syntactic and semantic constraints in a single grammar (e.g., Burton [1976]), or writing ad hoc programs that combine such information in unformalized ways.

There are some compensating disadvantages if the semantic domain is more complex than the syntactic one, but we will assume here that immediate semantic feedback is desired.

The first approach requires as many syntactic rules as semantic rules, and hence is not really much different from the semantic grammar approach (this is the conventional way of defining semantics of programming languages). The second approach has the tendency to miss generalities and its results do not automatically extend to new domains. It misses syntactic generalities, for example, by having to duplicate the syntactic information necessary to characterize the determiner structures of noun phrases for each of the different semantic kinds of noun phrase that can be accepted. Likewise, it tends to miss semantic generalizations by repeating the same semantic tests in various places in the grammar when a given semantic constituent can occur in various places in a sentence. The third approach, of course, may yield some level of operational system, but does not usually shed any light on how such interaction should be organized, and is difficult to extend.

Rusty Bobrow's RUS parser [Bobrow, 1978] is the first parser to my knowledge to make a clean separation between syntactic and semantic specification while gaining the benefit of early and incremental semantic filtering and maintaining the factoring advantages of an ATN. It's operation can be characterized by a generalization of ATN grammars that I have called cascaded ATN's (CATN's). A cascade of ATN's provides a way to reduce having to say the same thing multiple times or in multiple places, while providing efficiency comparable to a "semantic" grammar and at the same time maintaining a clean separation between syntactic and semantic levels of description. It is essentially a mechanism for permitting decomposition of an ATN grammar into an assembly of cooperating ATN's, each with its own characteristic domain of responsibility.

A CATN is essentially a sequence of ATN transducers with each successive machine taking input from the output of the previous one. Specifically, a CATN is a sequence of ordinary ATN's that include among the actions on their arcs an operation TRANSMIT, which transmits an element to the next machine in the sequence. The first machine in the cascade takes its input from the input sequence, and subsequent machines take their input from the TRANSMIT commands of the previous ones. The output of the final machine in the cascade is the output of the machine as a whole. The only feedback from later stages to earlier ones is a filtering function that causes paths of the nondeterministic computation to die if a later stage cannot accept the output of an earlier one.

The conception of cascaded ATN's arose from observing the interaction between the lexical retrieval component and the "pragmatic" grammar of the HWIM speech understanding system [Woods et al., 1976]. The lexical retrieval component made use of a network that consumed successive phonemes from the output of an acoustic phonetic recognizer and grouped them into words. Because of phonological effects across word boundaries, this network could consume several phonemes that were part of the transition into the next word before determining that a given word was possibly present. At certain points, it would return a found word together with a node in the network at which matching should begin to find the next word (essentially a state remembering how much of the next word has already been consumed due to the phonological word boundary effect). This can be viewed as an ATN that consumes phonemes and transmits words as soon as its has enough evidence that the word is there.

The lexical retrieval component of HWIM can thus be viewed as an ATN whose output drives another ATN. This led to the conception of a complete speech understanding system as a cascade of ATN's, one for acoustic phonetic recognition, one for lexical retrieval (word recognition), one for syntax one for semantics, and one for subsequent discourse tracking. A predecessor of the RUS parser [Bobrow, 1978] was subsequently perceived to be an instance of a syntax/semantics cascade, since the structures that it was obtaining from the lexicon to filter the paths through the grammar could be viewed as ATN's. Hence, practical solutions to problems of combinatorics in two different problem areas have independently motivated computation structures that can be viewed as cascaded

ATN's. It remains to be seen how effectively cascades can be used to model acoustic phonetic recognition or to track discourse structure, but the possibilities are intriguing.

## 4.1 Specification of a CATN Computation

As with ordinary ATN's and other formal automata, the specification of the computation of a CATN will consist of the specification of an instantaneous "configuration" of the automaton and the specification of a transition function that computes possible successor configurations for any given configuration. Since CATN's are nondeterministic, a given configuration can in general have more than one successor configuration and may occasionally have no successor. One way to implement a parser for CATN's would be to explicitly mimic this formal specification by implementing the configurations as data structures and writing a program to implement the transition function. Just as for ordinary ATN's, however, there are also many other ways to organize a parser, with various efficiency tradeoffs.

A configuration of a CATN consists of a vector of state configurations of the successive machines, plus a pointer to the input string where the first machine is about to take input. The transition function (nondeterministic) operates as follows:

1. A distinguished register C is set (possibly nondeterministically) to the next input element to be consumed and the pointer in the input string following C is computed. Then a stage counter k is set to 1.

2. The state of the kth machine in the sequence is used to determine a set of arcs that may consume the current input (possibly following a sequence of JUMPs, PUSHes, and POPs to reach a consuming transition).

3. Whenever a transmission operation TRANSMIT is executed by the stage k machine, the stage k+1 configuration is activated to process that input, and the stage k+1 component of the configuration vector is updated accordingly. If the k+1 stage cannot accept the transmitted structure, the configuration is aborted.

As for a conventional ATN, the format of the state configurations of the individual machines consist of a state name, a set of registers and contents, and a stack pointer (or its equivalent).* Each element of a stack is a pair consisting of a PUSH arc and a set of register contents. Transitions within a single stage are the same as for ordinary ATN's.

## 4.2 Uses of CATN's

A good illustrative example of the use of cascaded ATN's for natural language understanding would be a three stage machine consisting of a first stage that performs lexical analysis, a second stage for syntactic analysis, and a third stage for semantic

* For example, Earley's algorithm for context free grammars [Earley, 1968] replaces the stack pointer with a pointer to a place where the configuration(s) that caused the push can be found. A similar technique can be used with ATN grammars.

analysis. The lexical stage ATN would consume letters from an input sequence and perform word identification, including inflectional analysis, decomposition of contractions, and aggregation of compound phrases, producing as its output a sequence of words with syntactic categories and feature values. This machine could also perform certain standard bottom-up, locally determined parsings such as constructing noun phrase structures for proper nouns and pronouns. Ambiguity in syntactic class, in word grouping, and in homographs within a syntactic class can all be taken care of by the non-determinism of this first stage machine (e.g., "saw" as a past tense of "see" vs present tense of "saw" can be treated by two different alternative outputs of this first stage machine).

This first stage machine is not likely to involve any recursion, unlike other stages of the machine, but does use its registers to perform a certain amount of buffering before deciding what to transmit to the next stage. Because machines such as this one will reach states where they have essentially finished with a particular construction and are ready to begin a new one, a convenient action to have available on their arcs is one to reset all or a specified set of registers to their initial empty values again. Such register clearing is similar to that which happens on a push to a lower level, except that here the previous values need not be saved. The use of a register clearing action thus has the desired effect without the expense of a push.

The second stage machine in our example will perform the normal phrase grouping functions of a syntactic grammar and produce TRANSMIT commands when it has identified constituents that are serving specific syntactic roles. The third stage machine will consume such constituents and incorporate them into an incremental interpretation of the utterance (and may also produce differential likelihoods for alternative interpretations depending on the semantic and pragmatic consistency and plausibility of the partial interpretation).

The advantage of having a separate stage for the semantic interpretation, in addition to providing a clean separation between syntactic and semantic levels of description and a more domain-independent syntactic level, is that during the computation, different partial semantic interpretations that have the same initial syntactic structure share the same syntactic processing. In a single "semantic" ATN, such different semantic interpretation possibilities would have to make their own separate syntactic/semantic predictions with no sharing of the syntactic commonality between those predictions. Cascaded ATN'S avoid this while retaining the benefit of strong semantic constraint.

## 4.3 Benefits of CATN's

The decomposition of a natural language analyzer into a cascade of ATN's gains a "factoring" advantage similar to that which ATN's themselves provide with respect to ordinary phrase structure grammars. Specifically, the cascading allows alternative configurations in the later stages of the cascade to share common processing in the earlier stages that would otherwise have to be done independently. That is, if several semantic hypotheses can use a certain kind of constituent at a given place, there need be only one syntactic process to recognize it.*

Cascades also provide a simpler overall description of the acceptable input sequences than a single monolithic ATN that combined all of the information into a single network would give. That is, if any semantic level process can use a certain kind of constituent at a given place, then there need be only one place in the syntactic stage ATN that will recognize it. Conversely, if

* One might ask at this point whether there are situations in which one cannot tell what is present locally without "top-down" guidance from later stages. In fact, any such later stage guidance can be implemented by semantic filtering of syntactic possibilities. For example, if there is a given semantic context that permits a constituent construction that is otherwise not legal, one can still put the recognition transitions for that construction into the syntactic ATN with an action on the first transition to check compatibility with later stage expectations (e.g., by transmitting a flag indicating that it is about to try to recognize this special construction).

there are several syntactic contexts in which a constituent filling a given semantic role can be found, there need be only one place in the semantic ATN to receive that role. (A single network covering the same facts would be expected to have a number of states on the order of the product, rather than the sum, of the numbers of states in the individual stages of the cascade.)

An additional advantage provided by the factoring commonality introduced by the cascade is that the resulting localization of early stage activities in a single place provides a single place for a given linguistic fact to be learned, rather than independent versions of essentially the same fact having to be learned in different semantic contexts. Moreover, the separation of the stages of the cascade provides a decomposition of the overall problem into individually learnable skills. These facts may be of significance not only for theories of human language development and use, but also for computer systems that can be easily debugged and can contribute to their own acquisition of improved language skill. The above facts suggest that the traditional characterization of natural language in terms of the levels of phonemes, syllables, words, phrases, sentences, and higher level pragmatic constructs may be more deeply significant than just a convenience for scientific manipulation.

## 4.4 Parsing with CATN's

Conceptually, each ATN in a cascade produces (nondeterministically) a sequence of inputs for the next stage, which the next stage then parses. One could implement a computer parsing algorithm for a cascade in several ways. For example, the individual components of a configuration could be incremented as described above, with the later stages advanced as soon as the earlier stages transmit something. Alternatively, the later stages could wait until the earlier stages have completed a path through the input sequence before they begin to process the output of the earlier stages. The latter approach has the advantage of not performing second stage analysis on a path that will eventually fail at the first stage. On the other hand, it will result in the first stage occasionally continuing to extend partial paths that could already be rejected at the second stage.

In general, one can envisage an implementation in which the second stage can wait until the first stage has proceeded some distance past the current point before commencing its operations. This could either be done by having a fixed "lookahead" parameter which would always run the first stage some number of transmissions ahead of the second stage, or one could have a command that the first stage could execute when it considered its current path sufficiently likely to make it worthwhile for the second stage to operate on it. In fact, to handle both of these cases, one could simply have the first stage buffer its information in registers until it is ready for the next stage to work on it and only then perform the transmissions. For the remainder of this paper, we will assume that this is done and that the next stage begins to operate as soon as its input is transmitted.

As presented above, an instantaneous configuration of a CATN is essentially a vector of configurations (let us call them IC's) for the individual stages of the cascade. However, since any two configuration vectors having the same IC in some component will perform the same computation for that component and will only differ when they transmit to a subsequent stage, a parsing implementation should merge such common components and only perform their processing once. This can be achieved by representing the set of instantaneous configurations of the CATN not simply as a set of IC vectors, but as a tree structure (TC) that merges the common initial parts of those vectors. That is, each vector representing an instantaneous configuration of the CATN will be represented by a path through the TC from root to leaf, with the successive nodes in the path being the successive IC's of the vector. It is straightforward to transform the transition function that computes successor configuration vectors from a given one into a transition function that computes successor TC's from a given TC.

The TC representation has the characteristic that as long as the common left parts of configuration vectors are merged, the computation of a given IC at some level k will be done only once. To fully capitalize on the factoring advantages of this representation, one would like to assure that the common initial parts of alternative configuration vectors remain merged. This happens automatically for alternative stage k+1 computations that stem from a common stage k configuration. However, it is possible for two distinct k stage configurations, which have gone their separate ways and accumulated their own trees of higher level configurations, to come again to essentially the same k-stage configuration via different paths. This can happen especially with lexical stage computations when one word is recognized and the parsing of the next word begins. To provide maximum factoring, it is thus necessary to check for such cases and merge subtrees when the IC's at their heads are found to be equivalent.

When the k-stage network happens to be a finite state machine (i.e., makes no use of registers or recursion) the detection of a duplicate configuration is easy due to the simple equivalence test (i.e., sameness of state). When it is a general ATN, the detection of the conditions for merging are somewhat more involved (due to the register contents), and the likelihood of such merging being possible tends to be less. Hence for such stages the cost of checking for duplication may not be worth the benefit. Interestingly, it appears that the early stages of phonetic, lexical, and simple phrase recognition do have essentially finite state transition networks, while those of the later stages, where such sharing is not as important or as likely, is more apt to require non-finite-state register activities.

## 4.5 Comparison of Cascading with Recursion

Some interesting questions arise when considering the nature of cascaded ATN's as automata. For example, since a number of activities that are normally done with recursion in ATN's and other phrase structure grammars can be done by separate stages of a cascade, one is led to wonder about the relationship between cascading and recursion. That is, instead of arcs of an ATN pushing for a constituent of a certain kind, occasionally a cascade can be set up to find constituents of that kind and transmit them to a later stage of the cascade as units. A particular example, which has occasionally been proposed informally, would be for an early stage processor to group the input words into basic noun phrases, verb groups, etc. and for a later stage to take such units as input. Clearly this is a task normally performed by recursion. One might then wonder whether cascading was just another form of recursion, or somehow equivalent to it.

It turns out that cascading is in some respects weaker than recursion, and in other respects it is more powerful. In the next section, I will give an example of a context free cascade that can recognize a language that cannot be recognized by a single context free ATN. Hence, cascading clearly increases the power of a basic ATN beyond that provided by recursion alone. On the other hand, one is considerably more constrained in the way he can use cascading when writing a grammar than he is in the use of recursion. For example, indefinitely deep recursion can be used to recognize noun phrases inside prepositional phrases inside noun phrases, etc. When setting up a cascade of two ATN's to perform such grouping, the earlier cascade cannot model this directly, but instead would have to recognize "elementary" noun phrases consisting of, say, determiner, adjectives, and head noun, and would use looping transitions to accept subsequent prepositional phrases and relative clauses. Moreover, this stage of the cascade could not content itself solely with the noun phrases, but would also have to transmit the other elements of the sentence (auxiliaries, verbs, adverbs, particles, etc.) so that the later stages of the cascade will have a chance to see them. That is, a stage of a cascade provides a level of description of the entire input sequence in terms of a sequence of units to be transmitted to a later stage of analysis. Hence it appears that cascading is a fundamentally different operation that interacts with recursion and overlaps some of its functions in interesting ways.

Another interesting comparison arises between cascaded ATN's and the kinds of transformations used in a transformational grammar. If one were to attempt to analyze a transformational grammar by successively applying its transformations in reverse to the surface string, one repeatedly performs a partitioning of the input into a sequence of units as described above. That is, in applying a reverse transformation to a syntax tree in the course of a reverse transformational analysis, the operation of matching the pattern description of the transformation to the syntax tree amounts to finding a level at which the syntax tree can be "cut" yielding a sequence of units matching the sequence of elements in the pattern of the rule. This is exactly the kind of partitioning of the input into units that is done by a stage of a cascaded ATN. Moreover, the result of the transformation is expressed by a "right-hand-side" of the transformational rule, which may reorder the input sequence into a slightly modified sequence, and may copy an element several times, modify it in certain restricted ways, or even delete it (under suitable restrictions). In exactly the same way, a stage of a cascade can transmit the units that it has picked up in a different order than it found them, can duplicate a unit, drop a unit, insert a constant, and transmit units that are modified from the form in which they were recognized. In short, a stage of an ATN cascade can mirror the activity of any given transformational rule.

However, transformational rules are normally considered to apply in a cycle governed by the number of levels of embedding of clauses in the sentence, so that the number of successive transformations applied can be unbounded. By contrast, in an ATN cascade, there are only a finite number of stages in the cascade. Moreover, successive transformations in a transformational grammar are free to discard everything that was learned about the structure of the input in the matching of the previous transformation and there is no constraint that the manner in which a subsequent transformation analyzes the result of the previous transformation bear any relationship to the level of description imposed on the input by that previous transformation. In an ATN cascade, there is an assumed sequence of **progressive aggregation and higher level of description** implied by the transduction of information to successive stages of the cascade, with each stage perceiving the input in the terms that it was described by the previous. Thus, the ATN cascade seems to impose additional constraints on the process of language recognition that are not imposed by an ordinary transformational grammar.*

Experience with ATN grammars for natural language indicates that everything that a transformational grammar of natural language

* These constraints tend to promote the efficiency of the processing. See Woods [1970] for a discussion of some of the inherent inefficiencies of an ordinary transformational analysis.

does can be done with even a single ATN, so there does not appear to be any need for more than a finite number of stages of a cascade. On the other hand, the arguments presented here indicate that one may be able to obtain a simpler description of an overall set of facts with a cascade than with a single monolithic ATN. It is possible, therefore, that a cascade of ATN's corresponds to a more appropriate formalization of the underlying facts of language that gave rise to the original model of transformational grammar than does the conventional conception.

## 4.6 A Simple Formal Example

As a simple example of what a cascade of ATN's can do, I will give here a simple ATN cascade that without the use of registers can recognize the set of strings of the form n a's followed by n b's followed by n c's, for arbitrary n. This language is a traditional example of a language that is not context free but is context sensitive. However, it does happen to be specifiable as the intersection of two context free languages. Capitalizing on this fact, it is possible to represent it by a cascade of two "context free" ATN's (i.e., ATN's which do not use registers to check constraints between different constituents). This cascade effectively computes the intersection of two ways of viewing the input. The two ATN's, whose structure is illustrated in figure 1, can be written as follows:

```
(m1 (accepts q)
    (s1 (initial p q)
        ('a s2))
    (s2
        (p s3)
        ('b s4 (transmit 'b))
    (s3
        ('b s4 (transmit 'b))
    (s4 (pop p)
        ('c s5 (transmit 'c)))
    (s5 (pop q)
        ('c s5 (transmit 'c))))

(m2 (accepts r)
    (s1 (initial r)
        ('b s2))
    (s2
        (r s3)
        ('c s4))
    (s3
        ('c s4))
    (s4 (pop r)))
```

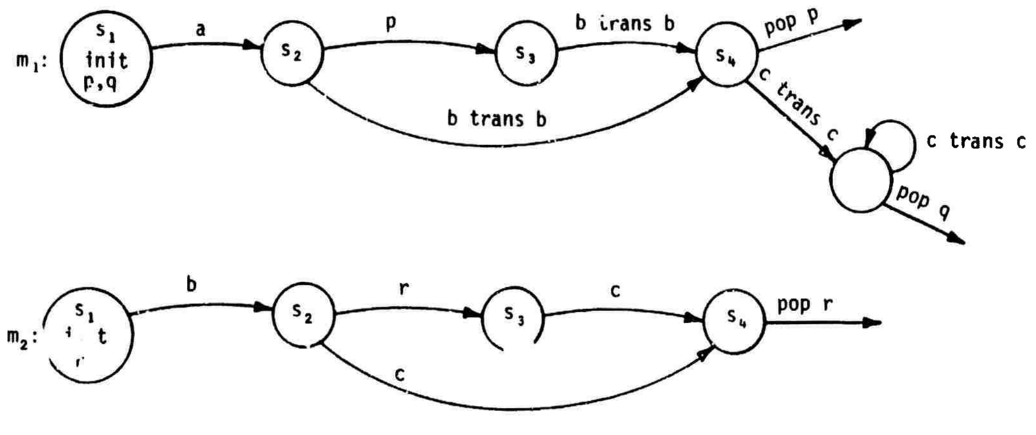

Fig. 1. ATN Cascade for $\{a^n b^n c^n : n \geq 1\}$

These two machines correspond to the grammars:

```
  q->pc*, p->ab, p->apb
and
  r->bc, r->brc
```

with augmentation such that the b's and c's accepted by the first grammar are passed through to be accepted by the second. The first stage checks that the number of a's and b's agree and accepts any number of c's, while the second stage requires that the b's and c's agree.

## 4.7 Another Example - Syntax and Semantics

Another, less trivial example is the use of an ATN cascade to represent syntactic and semantic knowledge sources of a language understanding system. We will give here a brief example illustrating a kind of cascading of syntactic and semantic knowledge similar to that done by R. Bobrow in his RUS parser [Bobrow, 1978]. A rough characterization of this parser is that as the syntactic component works its way through a noun phrase, it accumulates information about the determiner structure and initial premodifiers of the head noun until it encounters the head noun (i.e., takes a path corresponding to a hypothesis that it has found the head noun). At that point, it begins to transmit information to the semantic stage, starting with the head noun, and followed by the premodifiers of that noun. Then it continues to pick up post modifiers of the noun phrase, transmitting them to the semantic stage as it encounters them, and finally, when it hypothesizes that the noun phrase is completed, it transmits the determiner information.

In a similar way, in the parsing of a clause, the syntactic ATN can wait until it has encountered the main verb before transmitting that verb followed by its subject and any fronted adverbial modifiers. After that it can transmit subsequent post verbal elements as they are encountered, and finally transmit any governing modality information such as tense, aspect, and any governing negations.

The example presented here, is a constructed one to illustrate the principle, and does not directly represent the analyses by the RUS grammar. The example implements a subset of the semantic rules of the airline flight schedules system of Woods [1967], a predecessor of the LUNAR system [Woods et al.,1972]. I will give here only a fragment of the semantic stage ATN that understands designators (i.e., noun phrases). It assumes that the syntactic stage operates as outlined above and, in particular, that it transmits prepositional phrases by transmitting the preposition and then transmitting its object. It also assumes that the syntax stage transmits a signal QUANT when it has hypothesized the end of a noun phrase and is about to transmit the determiner and number information. One could alternatively transmit prepositional phrases as single units to be tested for syntactic and semantic features. I will assume that a pattern such as <flight> on a consuming arc is matched by a constituent that receives the indicated semantic marker (e.g., FLIGHT).

```
(m2 (accepts designators)
  (dl (initial designator)
    (J d2 (setr vbl (getnewvar)))
  (d2
    ('flight,'plane d/flight (setr head 'FLIGHT))
    ('jet d/flight (setr head 'FLIGHT)(addr mods '(JET !vbl)))
    ('airline d/head (setr head 'AIRLINE))
    ('city,'town d/head (setr head 'CITY))
    ('airport,'place d/head (setr head 'AIRPORT))
    ('time d/time)
    ('fare d/fare)
    ('owner,'operator d/owner))
  (d/owner
    ('of d/owner-of))
  (d/owner-of
    (<flight> d/head (addr quants (getquant !c))
       (setr head '(OWNER !c))))
  (d/fare
    ('(mod first-class),'(mod coach),'(mod stand by) d/fare
      (require (not class))
      (setr class !c))
    ('(mod one-way),'(mod round-trip) d/fare
      (require (not type))
      (setr type !c)))
    ('from d/fare-from (require (not from)))
    ('to d/fare-to (require (not to)))
    (J d/head (require class type from to)
      (setr head '(FARE !from !to !type !class)))
  (d/fare-from
    (<place> d/fare (addr quants (getquant !c)) (setr from !c)))
  (d/fare-to
    (<place> d/fare (addr quants (getquant !c)) (setr to !c)))
  (d/time
    ('(mod departure) d/time (require (not op)) (setr op 'DTIME))
    ('(mod arrival) d/time (require (not op)) (setr op 'ATIME))
    ('of d/time-of (require (not flight)))
    ('in,'at d/time-prep (require (eq op 'ATIME)))
    ('from d/time-prep (require (eq op 'DTIME)))
    (J d/head (require op flight place) (setr head '(!op !flight !c))
                        (*e.g., (setr head ' (ATIME AA-57 CHICAGO)))))
```

```
(d/time-of
  (<flight> d/time (addr quants (getquant !c)) (setr flight !c)))
(d/time-prep
  (<place> d/time (addr quants (getquant !c)) (setr place !c)))

(d/head
   ('QUANT d/quant (setr mod !(packmods))))
(d/flight
  ('(mod non-stop) d/non-stop-flight)
  ('from d/flight-from (require (not from)))
  ('to d/flight-to (require (not to)))
  ('(mod first-class),'(mod coach),'(mod jet-coach) d/flight
    (once class) (addr mods '(SERVCLASS !vbl !c)))
  ('(mod jet) d/flight (addr mods '(JET !vbl)))
  ('(mod propeller) d/flight (once equip)
       (addr mods '(NOT (JET !vbl))))
  (J d/flight (once connect) (require from to)
    (addr mods '(CONNECT !vbl !(sem from) !(sem to))))
  ('QUANT d/quant (setr mod !(packmods))))

(d/flight-from
  (<place> d/flight
         (addr quants (getquant !c))
         (setr from !c))

(d/flight-to
   (<place> d/flight
         (addr quants (getquant !c))
         (setr to !c)))
(d/quant
  ('some,'a,'any,'NIL d/some)
  ('each,'every d/each)
  ('all d/all)
  ('not d/not)
  ('the d/the)
  ('this,'that d/this)
  ('which,'what d/what)
  (<integer> d/integer))
(d/some
  ('sg,'pl d/end
    (setr quant '(FOR SOME !vbl / !head : !mod ; DLT))))
(d/each
  ('sg d/universal))
(d/all
  ('pl d/universal))
(d/universal
      (J d/end (setr quant '(FOR EVERY !vbl / !head : !mod ; DLT))))
    (d/not
      ('some d/not-some)
      ('every d/not-every)
      ('all d/not-all))
    (d/not-some
      ('sg,'pl d/end
        (setr quant '(NOT (FOR SOME !vbl / !head : !mod ; DLT)))))
    (d/not-every
      ('sg d/not-universal))
    (d/not-all
      ('pl d/not-universal))
    (d/not-universal
      (J d/end
        (etr quant '(NOT (FOR EVERY !vbl / !head : !mod ; DLT)))))
    (d/the
      ('sg d/end (setr quant '(FOR THE !vbl / !head : !mod ; DLT)))
      ('pl d/end (setr quant '(FOR EVERY !vbl / !head : !mod ; DLT))))
    (d/this
      ('sg d/end (setr quant '(FOR THE !vbl / !head : !mod ; DLT))))
    (d/what
      ('sg d/end (setr quant
        '(FOR THE !vbl / !head : (AND !mod DLT) ; (PRINTOUT !vbl))))
      ('pl d/end (setr quant
        '(FOR EVERY !vbl / !head : (AND !mod DLT) ; (PRINTOUT !vbl)))))
    (d/integer
      ('sg,'pl d/end (setr quant
        '(FOR !integer MANY !vbl / !head : !mod ; DLT))))

    (d/end
      (pop <designator> (sem-quant !quants !quant !vbl))))
```

In the above fragment grammar, the state d1 gets a variable name to use for the recognized designator, the state d2 dispatches on the head noun of the designator phrase to various states that recognize modifiers that are particular to the head. Eventually the path for each such head will lead to the state d/quant, where the determiner and number information is picked up to build the quantifier that governs this designator. This transition is triggered by the transmission of the flag QUANT from the syntax stage, signalling that the noun phrase is complete and the determiner information is coming. Notice how the quantification information that is common to most designators is shared.

The transitions that follow d/quant implement most of the d-rules in Woods [1967], which is itself a subset of the d-rules of the LUNAR system [Woods, et al., 1972; Woods, 1978b]. The function sem-quant is a function that performs the sem-quant pair manipulations described in Woods [1978b]. These manipulations usually embed the quantifier just constructed (!quant) into the quantifier nest accumulated from below (!quants) to form a quantifier nest to be passed up to a higher clause. They then return the variable name (!vbl) as the "sem" to be inserted into an argument position in the higher structure. The function getquant, here, is a function that extracts the quant from a structure that has been passed up from below and is used to accumulate the quantifier nest (quants) from subordinate designators that should dominate the quantifier of the designator being interpreted. The function packmods examines the contents of the register mods and returns an AND of the mods if there are several, a single mod if there is only one, and T if there are none.

## 5. KLONE Networks as Transition Networks

As mentioned previously, an ATN as an abstract automaton is characterized by its set of states and rules of transition, and not necessarily by a particular surface syntax for specifying those states and transitions. In many situations, such as the semantic stage ATN above, because of regularities in the behavior of classes of states and transitions, it is attractive to develop a surface notation in which more of the state transition behavior is implicit, so that a grammar designer need not explicitly indicate transition behavior that is highly regular. This is especially true for grammar segments that are intended to pick up constituents that can occur in relatively arbitrary order. In this section, I will show how KLONE concepts can be interpreted as ATN's and thereby provide a syntax that promotes sharing of regular behavior.

ATN's provide two ways of dealing with unordered constituents. One is to construct a separate state for each of the situations that could occur in the process of accumulating those constituents, and the other is to use a single state with looping transitions to pick up each constituent while using the registers to keep track of which constituents have been found. The former results in fast operation at execution time, but a potentially combinatorial increase in the number of states that are explicitly enumerated at compile time. The latter results in a more compact grammar, with fewer states enumerated at compile time, but with additional computation performed at parse time to determine which arcs are possible as a function of information accumulated in the registers.

The use of flag registers and actions such as DEC, REQ, and ONCE to characterize such behavior permits a grammar designer to develop the more compact form of grammar representation, while permitting a compiler that reads in a grammar to make either decision about the actual implementation (i.e., it can construct a separate state for all realizable combinations of state and flag registers, or it can construct a configuration in which state and flag registers are distinct components, with fast tests to filter the arcs leaving a state according to their flag requirements). Hence, the self-looping realization with special flag registers appears to provide the level of representation that a grammar designer should look at and think in terms of, while leaving questions of implementation up to a later compiler.

The above situation is a lot like the situation of modeling a physical device such as an elevator as a finite state machine. One can think of an elevator as being modeled by a state determined by the floors that have made requests (for each direction), where the car is, what direction it is travelling, and whether the doors are open. However, one does not want to enumerate all combinations of these variables since the behavior exhibited is regular and can be componentially represented. Nevertheless, thinking of such a componential representation as an abbreviatory specification of a finite state machine is a useful way to characterize what the representation means and how the elevator's behavior is being modeled. In a similar way, ATN's can be used to understand systems of grammar rules whose notational structure specifies transition behavior componentially.

Given the above perspective, it useful to think of the concepts in a KLONE inheritance network as a syntactic abbreviatory device for an ATN. In particular, we can take the following view. A Concept node will correspond to a state like d/fare above. Each Role of the Concept (either directly present or inherited) will correspond to a loop transition that sets a register corresponding to that Role. If the Role has a number facet =1 then the setting is done with setr. If the Role has a number facet >= Ø then the setting is done with addr. Popping is governed by an implicit pop transition which has requirements that each Role with a necessary modality facet be filled. The constituent popped will have a syntactic category identical with the concept node name.

From this perspective, the d/fare state in the above example would have an equivalent KLONE Concept:

```
(fare)
  has Roles:
    head  'fare  necessary =1
    from <place>  necessary =1
    to   <place>  necessary =1
    type  (or 'one-way 'round-trip) necessary =1
    class (or 'coach 'first-class 'jet-coach) necessary =1
```

where the name of the Concept is "fare", the Roles are named "head", "from", "to", "type", and "class", with value restrictions and modalities as indicated.

The only thing missing from this specification that is present in the above ATN fragment is the characterization of the transformation that is to take place to construct the structure that is popped. That is, the above Concept characterizes in some sense the surface structure that the ATN would have recognized - what Roles were filled and what relationships they had to each other - but does not indicate, as the ATN could do, a transformation of this surface structure into an underlying semantic representation. This can be done in the KLONE formalism by means of an "interp cable" (see Woods [1978a]). An interp cable points to another KLONE concept and maps Roles from the source Concept into corresponding Roles (perhaps several levels deep) in the destination Concept. This corresponds to the form on the pop arc of an ATN that indicates how the structure to be returned is to be built up. Thus, the interp cables in the KLONE nets correspond in some sense to the pop transitions of the ATN.

## 5.1 Inheritance

The major advantage of going from a conventional ATN notation to a KLONE notation for the semantic interpretation stage of an ATN cascade comes from the introduction of the notion of inheritance that results. That is, it is possible in the KLONE notation to have several different levels of description of a structure, while sharing common information among them. Thus, for example, when a semantic stage transition network has consumed a head noun "man", it would like to inherit from the Concept <person> the arcs that accept modifiers that are unique to people and from the Concept <physical object> those modifiers that apply to physical objects in general. The KLONE network notation, with its explicit inheritance of Roles from higher Concepts, provides exactly the sort of structure that is needed. Moreover, the explicit linking of Roles as differentiations or restrictions of higher Roles permits sharing of certain actions that are regularly performed on many different arcs (e.g., the "(addr quants (getquant !c))" action that occurs on every arc that consumes a subordinate that might generate quantifiers).

The use of inheritance networks to specify ATN's as described here appears to be an attractive way to organize complex "grammars" with extensive componential sharing. However, at least in its present state of formalization, KLONE structures appear to characterize only a subset of the class of possible ATN's and do not seem to be a universal subset. Specifically, the description above gives a characterization of the kinds of things that are done in the example of d/fare where the arcs are essentially consuming successive constituents of a single structure. However, the self-looping structure of this particular example is not typical of the transition structure of ATN's in general. When the result of a transition is to shift the ATN to a new state rather than to return to the original state with one more slot filled, the corresponding operation in a KLONE structure is not obvious. One might possibly achieve it by adding some form of augmentation to the Role nodes in KLONE to indicate the state to which the transition should go, but in general, this would depend on the actual node at which the Role was inherited and perhaps the state of filling of the other Roles.

The above problem appears to be a general difficulty for adding mechanisms of inheritance to ATN grammars for other than self-loop transitions. For example, in the LUNAR parser, a mechanism of "active" LEXARCS in a dictionary entry permitted specified words to add arcs to the current state of the grammar, thus capturing the regularities of words such as "and" and "or", whose effect depended more on the word than on the state of the grammar. However, here also, one faced the problem of determining to what state the inserted arc should make its transition. In general, this requires a facility for asking what the current state is and computing an appropriate destination state from it. In actuality, the LEXARC feature was only used in LUNAR to invoke a special SYSCONJ arc for conjunctions and this arc behaved essentially like a very complicated self-loop.

## 6. Generalized Transition Networks

The above interpretation of KLONE nets as ATN's is one way of dealing with constituents whose relative order of occurrence is not critical. In those cases where relative order is important, a KLONE net Concept can explicitly indicate the ordering constraint as a Structural Description associated with those constituents. Thus, the KLONE net formalization appears to be able to deal with the accumulation of constituents of a concept with arbitrary degrees of constrained order. However, even in the above example, the constituents were still considered in a unique order that was determined by the order of their transmission from the earlier stage ATN. A more general notion of transition network grammar, which I will call a "generalized transition network" (GTN) removes this limitation and generalizes the hypothesis factoring advantages of ATN grammars to automata that parse arbitrarily structured collections of input.

The idea of a GTN stems from the following observation: In an ATN, the set of transitions leaving a given state of the network does double duty - both specifying the alternative possible next states that one can go to as a result of measuring additional information about the input utterance, and also specifying implicitly that the measurement is to be made immediately to the right of the previous measurement in the input string. Thus, in following a sequence of arcs through an ATN grammar, one is both following the sequence of tests and hypothesis refinements that go on in the process of recognition and also following the left-to-right sequence of constituents in the input sentence. For many potential situation recognition applications, unlike for sentences the input is not simply a linear sequence of symbols. In such cases, the ATN's characterization of a sequence of information gathering activities is still desirable even though the idea of a left-to-right sequence of constituents does not make sense.

A GTN provides the appropriate automaton for such applications by keeping the general state transition structure of an ATN, but removing the implicit assumptions about the kind and location of the information gathering operations that result in a transition. When following a sequence of arcs through a GTN, one will still be following a sequence of hypothesis refinement operations, but there will no longer by any implicit left-to-right assumptions about the successive measurements. Rather, explicit instructions on the arcs will indicate how successive measurements relate to previous ones. We will assume the structures being recognized satisfy a **generalized constituent structure constraint**; they consist of structural assemblies of constituents of specified types standing in specified relationships to each other, with each constituent type either a specified elementary constituent type or itself characterized as a structured assembly of other constituents. Note that this is exactly the class of structures that can be characterized by KLONE Concepts.

In a GTN, information about whether successively acquired pieces of information are right/left, above/below each other, etc. will be explicitly indicated by operations on the arcs of the network. We will assume that the input elements of information are associated with "addresses" in some systematic **indexing structure** such as left-to-right word order, two-dimensional retinal position, position of a sentence in an ongoing dialog, time index of a portion of a continuous speech signal, node position in an arbitrary graph structure, etc., and that there are operations available for testing relative positions of constituents in this structure. Moreover, we may require specific accessing functions that, given a position in the addressing structure, can access related positions or scan in specified "directions" from the specified position. A GTN which actively uses such accessing functions to explore its input will be referred to as an "active" parser, while one that responds to an input sequence of perceptions that is determined by an external agent will be called "passive". In either case, an input element will consist of a pairing of an elementary constituent with the positional address of that element in the indexing space.

Formally, one can specify a GTN by the following components:

1. **A perceptual domain** - the space of possible input structures to which the GTN parser will be applied. Examples include strings of symbols from a finite vocabulary (as in traditional formal language theory), two-dimensional arrays of light intensities or hue descriptors (as in image understanding), sequences of sentences paired with speakers and unordered collections of facts and beliefs (for dialog understanding), and continuously varying time functions over a finite time interval (e.g., energy, fundamental frequency, and formant positions for a speech signal). Individual perceptions from a perceptual domain will be assumed to be composed of "elements" standing in some relationship to each other.

The perceptual domain will be assumed to have an associated **indexing space**, with respect to which elements of a perception can be "located" on a "perceptual field". Examples of indexing spaces are serial position of elements in a string, coordinates of positions in a two-dimensional (or n-dimensional) array, time points and time intervals in continuous signals, and node names in arbitrary graph structures.

2. A set of **probing operations** or measurements that can be used on an arc to indicate the conditions under which a transition can be made. In general, these will permit a probe to be made at any point in the indexing space. A probing operation will be assumed to set a focus pointer to the location in the indexing space where the measurement was made, and subsequent probes can take this focus pointer as a point of departure. Different probing

operations may make measurements at the same location, leaving the focus unchanged, may shift the focus by a specified amount in a specified direction and make a measurement there, or may scan from the current focus in a specified "direction" (or by some other specification of a trajectory of successive focus locations) until some measurement predicate is satisfied (returning the value of that measurement and leaving the focus set at the point where the measurement occurred).

3. A **structural space** in which the descriptions of parsed or recognized inputs will be constructed.

4. A set of **constituent types** that can be recognized.

5. A distinguished **top-level constituent type**, specifying the kinds of constituents that can be taken as a characterization of the entire perceptual complex being parsed.

6. A set of **state names** characterizing the possible states of the (nondeterministic) automaton in the course of parsing.

7. An **initial state function**, assigning one or more initial states to each constituent type, characterizing the states which can begin the knowledge acquisition process for a constituent of that type. Note that a given state may be an initial state of several different constituent types - this is useful when two different constituent types have a significant overlap of initial measurements (whether or not they have any actual overlap in constituent membership).

8. A **final state function**, assigning one or more final states to each constituent type, characterizing the states of the automaton at which complete recognition of a constituent of that type could be signaled. Note again, that a given state could be a final state for several constituent types - this would occur when two or more constituent types had possible common members.

9. A set of **arcs** connecting pairs of states, consisting of one of the following types:

PROBE <measurement operation>, an arc that enables a transition if the indicated measurement operation returns a non-failure result and sets the focus to the location of that measurement.

JUMP, an arc that enables a transition from one state to another without a probing operation, although computations can be done by the transition and conditions on the arc may block it.

ACCEPT <constituent type> <location constraints>, an arc that enables a transition when a constituent of the indicated type has been found satisfying the indicated location constraints. (In certain "top-down" parsing algorithms, the location constraints may be used to actively initiate the initial states for the desired constituent with appropriate focus locations. In other parsing algorithms, they are used only as filtering constraints on constituents that are independently found "bottom up".)

FOCUS <focus specification>, an arc that enables the focus to be relocated - e.g., to a location that has been saved in a register.

10. A set of **registers** that can hold the intermediate results of computations and the measurements that have been accumulated by a sequence of transitions. Registers can contain arbitrary elements from the structural space (which should include the structural descriptions of elements from the perceptual domain) and can also hold elements from the indexing space (in order to remember previous focus positions for later use).

11. A set of **structure manipulation operations** that can be used to set registers to the value of the current constituent (i.e. the constituent that enabled the current transition), the value of the current focus, the value of another register, a structural combination of the contents of other registers, or a specified constant.

12. A set of **transition augments** that characterize associated conditions and actions that must be satisfied (or performed) in order to take a given transition. These will include register and flag setting operations, a "require" action that blocks the transition unless specified conditions are satisfied, and a "cover" action that specifies that a given element (or region) of the input is to be considered covered or consumed by this transition. (One of the requirements for a complete parsing is that the entire input be "covered", analogous to the requirement in parsing a sequence of symbols that the end of the sequence must be reached.) This formulation of covering as an independent operation that may or may not be performed by an arc permits arcs to look at regions of the input beyond the boundaries of the region that they are attempting to parse without thereby asserting that the pieces of evidence that they have considered in this way have been fully accounted for. This situation is analogous to the use of lookahead in ordinary ATN's and other sequential parsers. We assume that ACCEPT arcs automatically cover all of the input covered by the constituent they accept.

13. A **completion condition** for each state that can be a final state of a constituent. The completion condition characterizes conditions on the register contents in order for a complete instance of that constituent to be recognized. Note that a given state will have completion conditions for each constituent type for which it is a final state.

14. A **construction function** for each constituent type that will construct a structural representation of a parsed constituent from the contents of the registers when a complete constituent has been recognized.

An ordinary ATN is a GTN whose perceptual domain is strings of words from a vocabulary, whose probing and measuring operations are the operations of CAT and WRD arcs that determine whether the next input word is satisfactory, and whose structural space is the space of tree or list structures. Constituent types are the phrase types recognized, and states, arcs, and transition augments are the states, arcs, and the conditions and actions of the ATN. The CAT and WRD arcs of the ATN are PROBE arcs with the appropriate measurement operations, and they automatically cover the input that they consume and move the focus beyond it. PUSH arcs are ACCEPT arcs and their location constraints are temporal adjacency to the focus. The focus in this case is simply the current position indicator in the input string.

A more interesting example of a GTN would be a transition network to understand visual images on a two-dimensional input field. In such an application, the coordinates of the indexing space could include positional information on the orientation of the head and the eye, as well as position on the retina, and probing measurements would include eye movement and head movement as possible operations. The trajectories for such probes could be determined by global measurements in the retinal field, including detection of "interesting" events by peripheral vision, and such probing may involve scanning in a given direction until an event of a certain kind is found, rather then the expectation of an immediately adjacent probe as is customary in sentence recognition.

Another interesting example would be a "middle-out" parser for speech understanding that probes for words at stressed syllables and other prosodically marked locations and works out from such positions to fill in the gaps. Likewise, a GTN to perform the acoustic phonetic analysis of the input waveform would be an interesting application because of its continuous perceptual domain, as would applications to tactile understanding and spatial exploration.

## 6.1 Observations

Although the GTN is at this point merely an abstract formulation of a generalization of the essential hypothesis factoring techniques of ATN grammars, it appears to be a very attractive framework for a variety of perceptual situations. For example, the control of the combinatoric possibilities of explicit hypothesis enumeration is an attractive feature for the analysis of visual scenes. In this connection, it is interesting to note that an observed characteristic of human visual processing would be explained by the use of a GTN for visual scene interpretation. Specifically, the observation that there appears to be a characteristic signature of eye movements for recognizing a given scene on separate presentations (at least in laboratory situations [Noton & Stark, 1971]) would be predicted. That is, if visual scene recognition were governed by a GTN as outlined above, then one would expect that attributes of the scene would govern the sequence of probes performed on the input in order to recognize it, and that replication of the input stimuli would replicate this sequence. (One would also expect changes in situation involving such things as a priori expectations and peripheral vision would alter the selection of probes, so that this characteristic signature might disappear outside of a controlled environment.)

## 7. Conclusions

In Woods [1977, 1978c; Woods & Brachman, 1978], I discussed the general principle of hypothesis "factoring" - i.e., the coalescing of common parts of alternative hypotheses in such a way that an incremental hypothesis development and search algorithm does not need to individuate and consider separate hypotheses until sufficient information is present to make different predictions in the different cases. The most common example of factoring is the well-known device called "decision trees" in which a cascade of questions at nodes of a tree leads eventually to selection of a particular "leaf" of the tree without explicit comparison to each of the individual leaves. If the tree is balanced, then this leads to the selection of the desired individual leaf in log(n) tests rather than n tests, where n is the number of leaves of the tree. Another example of factoring is the mechanism in ATN grammars whereby common parts of different phrase structure rules are merged, thereby saving the redundant processing of common parts of alternative hypotheses.

One can think of an ATN as a generalization of the notion of decision tree to permit recursion, looping, register augmentation, and recombination of paths. In this paper, I have discussed some families of automata that are variations or generalizations of ATN's and which provide similar hypothesis factoring capabilities. These include ATN cascades (CATN's), which permit a decomposition of complex language understanding behavior into a sequence of cooperating ATN's with separate domains of responsibility; KLONE networks, which can be interpreted as a kind of ATN that permits capturing grammar regularities through inheritance and a natural specification of relatively unordered constituents; and generalized

ATN's (GTN's), which remove the implicit assumptions of linear sequence of inputs that characterizes ordinary ATN's.

A GTN has a structure similar to an ATN in terms of its appearance and flow of control, except that information is accessed in response to explicit probing operations in the grammar rather than coming from an implicit "next symbol" operation. The states in the GTN correspond to states of knowledge, and the sequence of transitions leading to a state correspond to the info. ation seeking operations that lead to that state of knowledge.

The culmination of the development presented here would be a cascade of GTN's represented in some form of structured inheritance network, to permit sharing through inheritance and representations at varying levels of generality. Research is currently under way exploring the use of such automata for several stages of natural language understanding. Much work remains to be done to refine the notions of such automata and to assess their utilities. To this end, the presentation here may stimulate additional work along these lines.

Of specific interest are two distinct notions of the concept of factoring that are beginning to emerge from such considerations. One, which I have called **hypothesis factoring**, provides a reduction through sharing in the number of distinct hypotheses that have to be explicitly considered during parsing. The other, which I will call **conceptual factoring**, provides a reduction through sharing in the number of times or places that a given fact or rule needs to be represented in a long-term conceptual structure (e.g., the grammar). The former promotes efficiency of "run-time" parsing, while the latter promotes efficiency of grammar maintenance and learning. In many cases conceptual factoring promotes hypothesis factoring, but this is not necessarily always the case.

## References

Bobrow, R.J. 1978. "The RUS System", in B.L. Webber and R.J. Bobrow, Research in Natural Language Understanding, Quarterly Technical Progress Report No. 3. BBN Report No. 3878, Bolt Beranek and Newman Inc., Cambridge, MA. July.

Brachman, R.J. 1978a. "Structured Inheritance Networks," in Woods and Brachman, 1978, Research in Natural Language Understanding, Quarterly Technical Progress Report No. 1, 1 September 1977 to 30 November 1977, BBN Report No. 3742, Bolt Beranek and Newman Inc., Cambridge, MA. January.

Brachman, R.J. 1978b. "A Structural Paradigm for Representing Knowledge," Ph.D. dissertation, Division of Engineering and Applied Physics, Harvard University. Also, BBN Report No. 3605, Bolt Beranek and Newman Inc., Cambridge, MA, May.

Burton, R.R. 1976. "Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems." BBN Report No. 3453, Bolt Beranek and Newman Inc., Cambridge, MA, December.

Earley, J. 1968. "An Efficient Context-free Parsing Algorithm." Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Marcus, M.P. 1972. "A Theory of Syntactic Recognition for Natural Language," Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, M.I.T., Cambridge, MA, February.

Noton, D. and Stark, L. 1971. "Eye Movement and Visual Perception," **Scientific American**, June.

Smith, B.C. 1978. "Levels, Layers, and Planes: The framework of a system of knowledge representation semantics." Master's thesis. Artificial Intelligence Laboratory, MIT, January.

Thompson, F.B. 1963. "The Semantic Interface in Man-Machine Communication," Report No. RM 63TMP-35, Tempo, General Electric Co., Santa Barbara, CA, September.

Van Lehn, K. 1978. **Determining the Scope of English Quantifiers.** MIT Artificial Intelligence Laboratory, Technical Report No. 483.

Webber, B.L. 1978. "A Formal Approach to Discourse Anaphora", Ph.D. thesis, Harvard University. Also BBN Report No. 3761, Bolt Beranek and Newman Inc., Cambridge, MA, May.

Woods, W.A. 1967. "Semantics for a Question-Answering System," Ph.D. thesis, Division of Engineering and Applied Physics, Harvard University. Also Report NSF-19, Harvard Computation Laboratory, September. (Available from NTIS as PB-176-548, and from Garland Publishing, Inc. as a volume in a new series: **Outstanding Dissertations in the Computer Sciences.**)

Woods, W.A. 1969. "Augmented Transition Networks for Natural Language Analysis," Report No. CS-1, Aiken Computation Laboratory, Harvard University, December. (Available from ERIC as ED-037-733; also from NTIS as Microfiche PB-203-527.)

Woods, W.A. 1970. "Transition Network Grammars for Natural Language Analysis," **CACM**, Vol. 13, No. 10, October.

Woods, W.A., R.M. Kaplan, and B.L. Nash-Webber 1972. "The Lunar Sciences Natural Language Information System: Final Report," BBN Report No. 2378, Bolt Beranek and Newman Inc., Cambridge, MA, June. (Available from NTIS as N72-28984.)

Woods, W.A., M. Bates, G. Brown, B. Bruce, C. Cook, J. Klovstad, J. Makhoul, B. Nash-Webber, R. Schwartz, J. Wolf, V. Zue. 1976. Speech Understanding Systems - Final Report, 30 October 1974 to 29 October 1976, BBN Report No. 3438, Vols. I-V, Bolt Beranek and Newman Inc., Cambridge, MA.

Woods, W.A. 1977. "Spinoffs From Speech Understanding Research," in Panel Session on Speech Understanding and AI, Proceedings of the 5th Int. Joint Conference on Artificial Intelligence, August 22-25, p. 972.

Woods, W.A. 1978a. Research in Natural Language Understanding, Quarterly Technical Progress Report No. 2, 1 December 1977 to 28 February 1978. BBN Report No. 3797, Bolt Beranek and Newman Inc., Cambridge, MA, April. (Available from NTIS as AD No. AO49781.)

Woods, W.A. 1978b. "Semantics and Quantification in Natural Language Question Answering," in **Advances in Computers**, Vol. 17. New York: Academic Press. (Also Report No. 3687, Bolt Beranek and Newman Inc.)

Woods, W.A. 1978c. "Taxonomic Lattice Structures for Situation Recognition," in TINLAP-2, Conference on Theoretical Issues in Natural Language Processing-2, University of Illinois at Urbana-Champaign, July 25-27.

Woods, W.A. and Brachman, R.J. 1978. Research in Natural Language Understanding, Quarterly Technical Progress Report No. 1, 1 September 1977 to 30 November 1977, BBN Report No. 3742, Bolt Beranek and Newman Inc., Cambridge, MA, January. (Now available from NTIS as AD No. A053958).

Official Distribution List
Contract N00014-77-C-0378

| | Copies |
|---|---|
| Defense Documentation Center<br>Cameron Station<br>Alexandria, VA 22314 | 12 |
| Office of Naval Research<br>Information Systems Program<br>Code 437<br>Arlington, VA 22217 | 2 |
| Office of Naval Research<br>Code 200<br>Arlington, VA 22217 | 1 |
| Office of Naval Research<br>Code 455<br>Arlington, VA 22217 | 1 |
| Office of Naval Research<br>Code 458<br>Arlington, VA 22217 | 1 |
| Office of Naval Research<br>Branch Office, Boston<br>495 Summer Street<br>Boston, MA 02210 | 1 |
| Office of Naval Research<br>Branch Office, Chicago<br>536 South Clark Street<br>Chicago, IL 60605 | 1 |
| Office of Naval Research<br>Branch Office, Pasadena<br>1030 East Green Street<br>Pasadena, CA 91106 | 1 |
| Office of Naval Research<br>New York Area Office<br>715 Broadway - 5th Floor<br>New York, NY 10003 | 1 |

|  |  |
|---|---|
| Naval Research Laboratory<br>Technical Information Division<br>Code 2627<br>Washington, D.C. 20380 | 6 |
| Naval Ocean Systems Center<br>Advanced Software Technology Division<br>Code 5200<br>San Diego, CA 92152 | 1 |
| Dr. A.L. Slafkosky<br>Scientific Advisor<br>Commandant of the Marine Corps (Code RD-1)<br>Washington, D.C. 20380 | 1 |
| Mr. E.H. Gleissner<br>Naval Ship Research & Development Center.<br>Computation & Mathematics Dept.<br>Bethesda, MD 20084 | 1 |
| Capt. Grace M. Hopper<br>NAICOM/MIS Planning Branch (OP-916D)<br>Office of Chief of Naval Operations<br>Washington, D.C. 20350 | 1 |
| Mr. Kin B. Thompson<br>NAVDAC 33<br>Washington Navy Yard<br>Washington, D.C. 20374 | 1 |
| Advanced Research Projects Agency<br>Information Processing Techniques<br>1400 Wilson Boulevard<br>Arlington, VA 22209 | 1 |
| Capt. Richard L. Martin, USN<br>Commanding Officer<br>USS Francis Marion (LPA-249)<br>FPO New York 09501 | 1 |